📌 4. Project Design Phase (Detailed)

The **Project Design Phase** defines how different components of the Freelancing MERN Application interact with each other from a structural, visual, and database point of view. This phase transforms the planning into a tangible blueprint, helping ensure all modules are logically connected, reusable, and scalable.

🧩 4.1 System Architecture

The application follows a **client-server architecture** where the **React frontend** interacts with the **Express backend** using secure, RESTful API calls. Data is stored and retrieved from **MongoDB Atlas**, a cloud-based NoSQL database.

🔧 Layers Overview

scss

Copy code

Frontend (React + Axios)

↓ API Requests

Backend (Express + Node.js)

↓ Mongoose

Database (MongoDB Atlas)

- **Frontend** handles UI rendering, form control, routing, and state management.
- **Backend** handles authentication, role verification, data logic, and API endpoints.
- **Database** stores user accounts, project data, bids, submissions, and feedback.

🧱 4.2 Component and Module Design

🔸 Frontend Modules (React Components)

Component	Description
Login.jsx / Register.jsx	Handles user authentication forms
Navbar.jsx	Shows role-based links (Client, Freelancer, Admin)
CreateProject.jsx	Form for clients to post projects

Component	Description
ProjectList.jsx	Displays all projects available for bidding
FreelancerProfile.jsx	Shows freelancer details and feedback
AdminPanel.jsx	Admin dashboard to view/block users
BidForm.jsx / BidList.jsx	Allows freelancers to place bids and clients to view bids

All components are designed as reusable and maintainable using **React functional components** and useEffect, useState hooks.

🔹 **Backend Modules (API + Middleware)**

Module	Description
authRoutes.js	Handles login, registration, and /auth/me
projectRoutes.js	Handles project creation, fetching, bidding, submission, feedback
adminRoutes.js	Handles admin user viewing/blocking
auth.js middleware	Validates JWT and attaches user data to requests
admin.js middleware	Ensures only admins can access admin routes

Each module is separated using **MVC architecture**:

- **Model**: Mongoose schemas
- **Controller**: Request/response logic
- **Routes**: Connects URLs to controllers
- **Middleware**: Handles security and access control

🗄 **4.3 Database Design (MongoDB)**

MongoDB is used for its **flexibility** in schema design and easy scaling. The schema includes four main collections:

📁 **User Schema**

js

Copy code

{

name: String,

email: String,

password: String (hashed),

role: String (client/freelancer/admin),

isBlocked: Boolean,

}

📂 **Project Schema**

js

Copy code

{

clientId: ObjectId,

title: String,

description: String,

category: String,

bids: [ObjectId],

selectedFreelancer: ObjectId,

workSubmitted: Boolean

}

📂 **Bid Schema**

js

Copy code

{

projectId: ObjectId,

freelancerId: ObjectId,

bidAmount: Number,

message: String

}

📁 Feedback Schema

```
js

Copy code

{

 projectId: ObjectId,

 clientId: ObjectId,

 freelancerId: ObjectId,

 rating: Number,

 comment: String

}
```

All data is normalized using **ObjectId references**, supporting Mongoose population where needed (e.g., showing freelancer names inside bids).

🗄️ 4.4 API Route Design

The backend exposes **RESTful APIs** for smooth interaction between frontend and backend. Here's an overview of the main endpoints:

🔐 Auth Routes

Endpoint	Method	Description
/api/auth/register	POST	Register new user
/api/auth/login	POST	Authenticate user and return JWT
/api/auth/me	GET	Fetch current logged-in user using token

📦 Project & Bid Routes

Endpoint	Method	Description
/api/projects	POST	Client posts a new project
/api/projects	GET	Fetch all projects
/api/projects/:id/bid	POST	Freelancer places a bid
/api/projects/:id/select	POST	Client selects a freelancer

Endpoint	Method	Description
/api/projects/:id/submit	POST	Freelancer submits work

⭐ Feedback Route

Endpoint	Method	Description
/api/projects/:id/feedback	POST	Client submits rating & review

🛡️ Admin Routes

Endpoint	Method	Description
/api/admin/users	GET	Admin views all users
/api/admin/block/:id	PUT	Block/unblock a user

All protected routes require a valid **JWT token** and, in some cases, an admin role validation.

👨‍🎨 4.5 UI/UX Design Principles

The frontend UI is built with **Tailwind CSS** for fast development and consistent styling. Key design features include:

- **Role-based navigation bar** showing relevant options (e.g., AdminPanel only for admins)
- **Clean forms** with validation messages
- **Mobile-first responsiveness**
- **Interactive project cards** and bid modals
- **Feedback display** with average ratings shown as stars
- **Admin Panel** with clean user tables and toggle buttons

All designs prioritize **clarity, minimalism, and usability** — ideal for academic and internship grading.

🧰 Reusability & Scalability Design

- **Reusability**: Components like ProjectCard, BidItem, and FeedbackBox can be reused across pages.
- **Scalability**: The system can easily add features like:

- Payment gateway (e.g., Razorpay/Stripe)
- Chat system (e.g., Socket.io)
- Notifications
- Resume upload & filtering

✅ This phase resulted in a complete blueprint of how the system would be implemented across code structure, APIs, and UI, enabling smooth development in the upcoming implementation phase.